Decision Problems for Additive Regular Functions

Rajeev Alur and Mukund Raghothaman

University of Pennsylvania
{alur, rmukund}@cis.upenn.edu

Abstract. Additive Cost Register Automata (ACRA) map strings to integers using a finite set of registers that are updated using assignments of the form "$x := y + c$" at every step. The corresponding class of *additive regular functions* has multiple equivalent characterizations, appealing closure properties, and a decidable equivalence problem. In this paper, we solve two decision problems for this model. First, we define the *register complexity* of an additive regular function to be the minimum number of registers that an ACRA needs to compute it. We characterize the register complexity by a necessary and sufficient condition regarding the largest subset of registers whose values can be made far apart from one another. We then use this condition to design a PSPACE algorithm to compute the register complexity of a given ACRA, and establish a matching lower bound. Our results also lead to a machine-independent characterization of the register complexity of additive regular functions. Second, we consider *two-player games over ACRAs*, where the objective of one of the players is to reach a target set while minimizing the cost. We show the corresponding decision problem to be EXPTIME-complete when costs are non-negative integers, but undecidable when costs are integers.

1 Introduction

Consider the following scenario: a customer frequents a coffee shop, and each time purchases a cup of coffee costing \$2. At any time, he may fill a survey, for which the store offers to give him a discount of \$1 for each of his purchases that month (including for purchases already made). We model this by the machine M_1 shown in figure 1.1. There are two states q_S and $q_{\neg S}$, indicating whether the customer has filled out the survey during the current month. There are three events to which the machine responds: C indicates the purchase of a cup of coffee, S indicates completion of the survey, and # indicates the end of a month. The registers x, y track how much money the customer owes the establishment: in state $q_{\neg S}$, the amount in x assumes that he will not fill out a survey that month, and the amount in y assumes that he will fill out a survey before the end of the month. At any time the customer wishes to settle his account, the machine outputs the amount of money owed, which is always the value in register x.

The automaton M_1 has a finite state space, and a finite set of integer-valued registers. On each transition, each register (say u) is updated by an expression

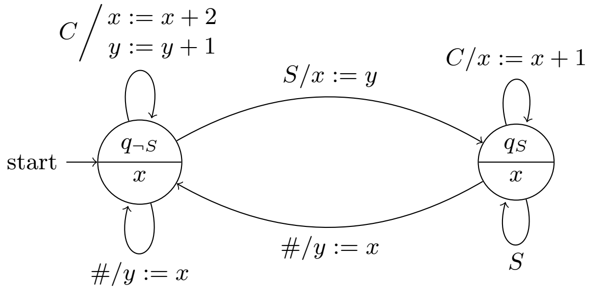


Fig. 1.1: ACRA M_1 models a customer in a coffee shop. It implements a function $f_1 : \{C, S, \#\}^* \to \mathbb{Z}$ mapping the purchase history of the customer to the amount he owes the store.

of the form "$u := v + c$", for some register v and constant $c \in \mathbb{Z}$. Which of these registers will eventually contribute to the output is determined by future events, and so the cost of an event depends not only on the past, but also on the future. Indeed, it can be shown that these machines are *closed under regular lookahead*, i.e. the register updates can be conditioned on regular properties of an as-yet-unseen suffix, for no gain in expressivity. The important limitation is that register updates are test-free, and cannot examine the register contents.

The motivation behind the model is generalizing the idea of regular languages to quantitative properties of strings. A language $L \subseteq \Sigma^*$ is regular when there is an accepting DFA. Regular languages are a robust class, permitting multiple equivalent representations as regular expressions and as formulas in monadic second-order logic. Recently in [3], we proposed the model of regular functions: they are the MSO-definable transductions from strings to expression trees over some pre-defined grammar. The class of functions thus defined depends on the grammar allowed; the simplest is when the underlying domain is the set of integers $\mathbb{Z}$, and expressions involve constants and binary addition, and we call these additive regular functions. Additive regular functions have appealing closure properties, such as under linear combination, input reversal, and regular lookahead, and several analysis problems are efficiently decidable – such as containment, shortest paths and equivalence checking. ACRAs correspond to this class of additive regular functions.

Observe that machine M_1 has two registers, and it is not immediately clear how (if it is even possible) to reduce this number. This is the first question that this paper settles: Given an Additive Cost Register Automaton (ACRA) M, how do we determine the minimum number of registers needed by any ACRA to compute $[\![M]\!]$? We describe a phenomenon called register separation, and show that any equivalent ACRA needs at least k registers iff the registers of M are k-separable. It turns out that the registers of M_1 are 2-separable, and hence two registers are necessary. We then go on to show that determining k-separability is

PSPACE-complete. Determining the register complexity is the natural analogue of the state minimization problem for DFAs [7].

The techniques used to analyse register complexity allow us to state a result similar to the pumping lemma for regular languages: The register complexity of f is at least k iff for some m, we have strings $\sigma_0, \ldots, \sigma_m, \tau_1, \ldots, \tau_m$, suffixes $w_1, \ldots, w_k$, and k distinct coefficient vectors $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{Z}^m$ so that for all vectors $\mathbf{x} \in \mathbb{N}^m$, $f\left(\sigma_0 \tau_1^{x_1} \sigma_1 \tau_2^{x_2} \ldots \sigma_m w_i\right) = \sum_j c_{ij} x_j + d_i$. Thus, depending on the suffix w_i, at least one of the cycles $\tau_1, \ldots, \tau_k$ contributes differently to the final cost.

Next, we consider ACRAs with turn-based alternation. These are games where several objective functions are simultaneously computed, but only one of these objectives will eventually contribute to the output, based on the actions of both the system and its environment. Alternating ACRAs are thus related to multi-objective games and Pareto optimization [15], but are a distinct model because each run evaluates to a single value. We study the reachability problem in ACRA games: Given a budget k, is there a strategy for the system to reach an accepting state with cost at most k? We show that this problem is EXPTIME-complete when the registers assume values from $\mathbb{N}$, and undecidable when the registers are integer-valued.

Related work The traditional model of string-to-number transducers has been (non-deterministic) weighted automata (WA). Additive regular functions are equivalent to unambiguous weighted automata over the tropical semiring, and are therefore strictly sandwiched between weighted automata and deterministic WAs in expressiveness. Deterministic WAs are ACRAs with one register, and algorithms exist to compute the *state complexity* and for minimization [13]. Mohri [14] presents a nice survey of the field. While the determinizability of weighted automata remains an open problem [9,4], it has been solved in polynomial time for the specific case of unambiguous weighted automata. There is a polynomial translation from unambiguous WAs to ACRAs, and the algorithm of subsection 4.1 runs in polynomial time when the number of registers $k = 2$. Thus, to the extent to which they are relevant, we match the bounds available in the literature. Recent work on the quantitative analysis of programs [6] also uses weighted automata, but does not deal with minimization or with notions of regularity. Data languages [8] are concerned with strings over a (possibly infinite) data domain $\mathbb{D}$. Recent models [5] have obtained Myhill-Nerode characterizations, and hence minimization algorithms, but the models are intended as acceptors, and not for computing more general functions. Turn-based weighted games [11] are ACRA games with a single register, and in this special setting, it is possible to solve non-negative optimal reachability in polynomial time. Of the techniques used in the paper, difference bound invariants are a standard tool. However when we need them, in section 3, we have to deal with disjunctions of such constraints, and show termination of invariant strengthening – to the best of our knowledge, the relevant problems have not been solved before.

Outline of the paper We define the automaton model in section 2. In section 3, we introduce the notion of separability, and establish its connection to register complexity. In section 4, we show that determining the register complexity is PSPACE-complete. Finally, in section 5, we study ACRA reachability games – in particular, that ACRA $(\mathbb{Z})$ games are undecidable, and that ACRA $(\mathbb{N})$ reachability games are EXPTIME-complete.

2 Additive Regular Functions

We will use additive cost register automata as the working definition of additive regular functions, i.e. a function[1] $f : \Sigma^* \to \mathbb{Z}_\perp$ is regular iff it is implemented by an ACRA. An ACRA is a deterministic finite state machine, supplemented by a finite number of integer-valued registers. Each transition specifies, for each register u, a test-free update of the form "$u := v + c$", for some register v, and constant $c \in \mathbb{Z}$. Accepting states are labelled with output expressions of the form "$v + c$".

Definition 1. *An ACRA is a tuple $M = (Q, \Sigma, V, \delta, \mu, q_0, F, \nu)$, where Q is a finite non-empty set of states, Σ is a finite input alphabet, V is a finite set of registers, $\delta : Q \times \Sigma \to Q$ is the state transition function, $\mu : Q \times \Sigma \times V \to V \times \mathbb{Z}$ is the register update function, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the non-empty set of accepting states, and $\nu : F \to V \times \mathbb{Z}$ is the output function.*

The configuration of the machine is a pair $\gamma = (q, val)$, where q is the current state, and $val : V \to \mathbb{Z}$ maps each register to its value. Define $(q, val) \to^a (q', val')$ iff $\delta(q, a) = q'$ and for each u, if $\mu(q, a, u) = (v, c)$, then $val'(u) = val(v) + c$.

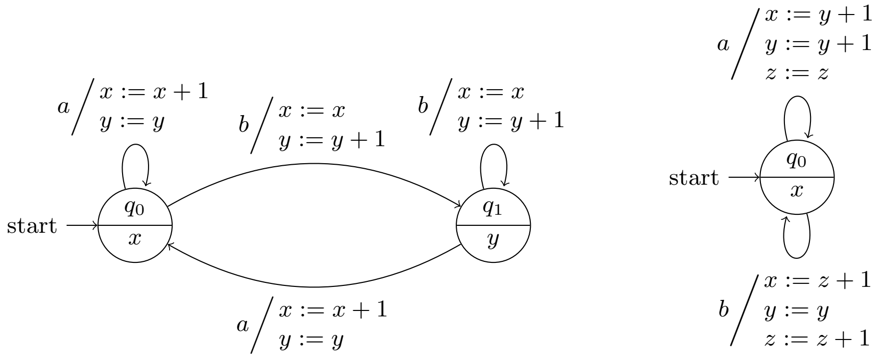
Machine M then implements a function $[\![M]\!] : \Sigma^* \to \mathbb{Z}_\perp$ defined as follows. For each $\sigma \in \Sigma^*$, let $(q_0, val_0) \to^\sigma (q_f, val_f)$, where $val_0(v) = 0$ for all v. If $q_f \in F$ and $\nu(q_f) = (v, c)$, then $[\![M]\!](\sigma) = val_f(v) + c$. Otherwise $[\![M]\!](\sigma) = \perp$.

We will write $val(u, \sigma)$ for the value of a register u after the machine has processed the string σ starting from the initial configuration.

Remark 1. Any given ACRA M can easily be *trimmed* so that every state q is reachable from the initial state. All claims made in this paper assume that the machines under consideration are trimmed.

An important precondition when we define k-separability will be that the registers be *live*. Informally, a register v is live in state q if for some suffix $\sigma \in \Sigma^*$, on processing σ starting q, the initial value of v is what influences the final output. For example, M_1 could be augmented with a third register z tracking the length of the string processed. However, the value of z would be irrelevant to the computation of f_1, and z would thus not be live. A straightforward way of defining live registers is through *suffix summaries*. Let q be a state, and $\sigma \in \Sigma^*$ be a string. Then the suffix summary of σ in q is either a register-offset pair

[1] By convention, we represent a partial function $f : A \to B$ as a total function $f : A \to B_\perp$, where $B_\perp = B \cup \{\perp\}$, and $\perp \notin B$ is the "undefined" value.



(a) M_2.

(b) M_3.

Fig. 2.1: ACRAs M_2 and M_3 operate over the input alphabet $\Sigma = \{a, b\}$. Both implement the function defined as $f_2(\epsilon) = 0$, and for all σ, $f_2(\sigma a) = |\sigma a|_a$, and $f_2(\sigma b) = |\sigma b|_b$. Here $|\sigma|_a$ is the number of occurrences of the symbol a in the string σ.

$V \times \mathbb{Z}$, or $\bot$, and which summarizes the effect of processing σ starting from state q. If the suffix summary of σ in q is (v, c), then it would be informally read as: "The result of processing suffix σ if the machine is currently in q is the current value of v plus c." Formally,

Definition 2. *Let q and q' be states so that $\delta(q, \sigma) = q'$.*

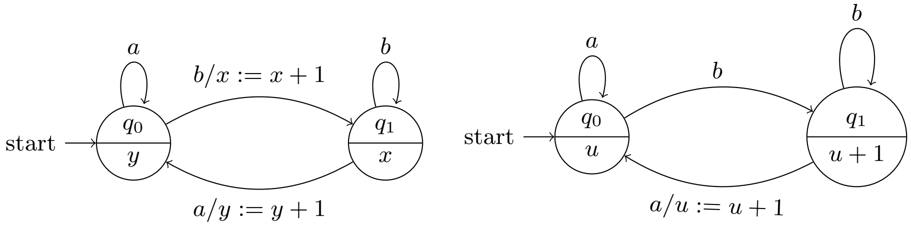
1. *If $q' \notin F$, then the suffix summary of σ in q is $\bot$, and*
2. *(otherwise if $q' \in F$) if $\nu(q') = (u, c)$, and $\mu(q, \sigma, u) = (v, c')$, then the suffix summary of σ in q is $(v, c + c')$.*

A register v is live *in a state q if for some $\sigma \in \Sigma^*$, $c \in \mathbb{Z}$, the suffix summary of σ in q is (v, c).*

Remark 2. Whether a register v is live in a state q is a static property of the state. At each state q, pick a register v_q which is live in q. If no such register exists, then arbitrarily choose $v_q \in V$. On all transitions into q, reset all non-live registers v to the value of v_q. This rewrite does not affect $[\![M]\!]$, and can be performed in linear time. All claims made in this paper assume that this rewrite has been performed.

We recall the following properties of ACRAs [3]:

Equivalent characterizations Additive regular functions are equivalent to unambiguous weighted automata [14] over the tropical semiring. These are non-deterministic machines with a single counter. Each transition increments the counter by an integer c, and accepting states have output increments, also integers.



(a) M_4.

(b) M_5.

Fig. 2.2: ACRAs M_4 and M_5 operate over $\Sigma = \{a, b\}$, and implement f_4 so that if σ ends in an a, then $f_4(\sigma)$ = number of a-s immediately following a b, and otherwise $f_4(\sigma)$ = number of b-s immediately following an a. When we omit the update for a register, say v, it is understood to mean, "$v := v$".

The unambiguous restriction requires that there be a single accepting path for each string in the domain, thus the "min" operation of the tropical semiring is unused. Consider the class of MSO-definable string-to-integer transductions, with the successor and predecessor operations allowed over integers. This class of functions coincides with additive regular functions. Recently, streaming tree transducers [2] have been proposed as the regular model for string-to-tree transducers – ACRAs are equivalent in expressiveness to regular string-to-term transducers with binary addition as the base grammar.

Closure properties What makes additive[2] regular functions interesting to study is their robustness to various manipulations:

1. for all $c \in \mathbb{Z}$, if f_1 and f_2 are regular functions, then so are $f_1 + f_2$ and cf_1,
2. if f is a regular function, then f_{rev} defined as $f_{rev}(\sigma) = f(\sigma^{rev})$ is also regular, and
3. if f_1 and f_2 are regular functions, and L is a regular language, then the function f defined as $f(\sigma) = \text{if } \sigma \in L, \text{ then } f_1(\sigma), \text{ else } f_2(\sigma)$ is also regular.
4. ACRAs are closed under regular lookahead, i.e. even if the machine were allowed to make decisions based on a regular property of the suffix rather than simply the next input symbol, there would be no increase in expressiveness.

Analysis problems Given ACRAs M_1 and M_2, equivalence-checking and the min-cost problem ($\min_{\sigma \in \Sigma^*} [\![M]\!](\sigma)$) can be solved in polynomial time. It follows then that containment (for all σ, $[\![M_1]\!](\sigma) \leq [\![M_2]\!](\sigma)$) also has a polynomial time algorithm.

[2] We will often drop the adjective "additive", and refer simply to regular functions.

3 Characterizing the Register Complexity

The register complexity of a function f is the minimum number of registers an ACRA needs to compute it. For example the register complexity of both $[\![M_1]\!]$ in figure 1.1 and $[\![M_2]\!]$ in figure 2.1a is 2, while the register complexity of $[\![M_4]\!]$ is 1. Computing the register complexity is the first problem we solve, and will occupy us for this section and the next.

Definition 3. *Let $f : \Sigma^* \to \mathbb{Z}_\perp$ be a regular function. The register complexity of f is the smallest number k so there is an ACRA M implementing f with only k registers.*

Informally, the registers of M are separable in some state q if their values can be pushed far apart. For example, consider the registers x, y of M_1 in state q_0. For any constant c, there is a string $\sigma = C^c$ leading to q_0 so that $|val\,(x, \sigma) - val\,(y, \sigma)| \geq c$.

Definition 4. *Let $M = (Q, \Sigma, V, \delta, \mu, q_0, \nu)$ be an ACRA. The registers of M are* k-separable *if there is some state q, and a collection $U \subseteq V$ so that*

1. *$|U| = k$, all registers $v \in U$ are live in q, and*
2. *for all $c \in \mathbb{Z}$, there is a string σ, so that $\delta\,(q_0, \sigma) = q$ and for all distinct $u, v \in U$, $|val\,(u, \sigma) - val\,(v, \sigma)| \geq c$.*

The registers of a machine M are not k-separable if at every state q, and collection U of k live registers, there is a constant c so for all strings σ to q, $|val\,(u, \sigma) - val\,(v, \sigma)| < c$, for some distinct $u, v \in U$. Note that the specific registers which are close may depend on σ. For example, in machine M_3 from figure 2.1b, if the last symbol was a, then x and y will be close, while if the last symbol was a b, then x and z are guaranteed to be equal.

Theorem 1. *Let $f : \Sigma^* \to \mathbb{Z}_\perp$ be a function defined by an ACRA M. Then the register complexity of f is at least k iff the registers of M are k-separable.*

The two directions of the proof are presented separately in the following subsections.

3.1 k-separability implies a lower bound on the register complexity

Consider machine M_1 from figure 1.1. Here $k = 2$, and registers x, y are separated in state $q_{\neg S}$. Let $\sigma_1 = \epsilon$, i.e. the empty string, and $\sigma_2 = S$ – these are suffixes which, when starting from $q_{\neg S}$, "extract" the values currently in x, y.

Now suppose an equivalent counter-example machine M' is proposed with only one register v. At each state q' of M', observe the "effect" of processing suffixes σ_1, σ_2. Each of these can be summarized by an expression of the form $v + c_{q'i}$ for $i \in \{1, 2\}$, the current value of register v, and $c_{q'i} \in \mathbb{Z}$. Thus, the outputs differ by no more than $|(v + c_{q'1}) - (v + c_{q'2})| \leq |c_{q'1}| + |c_{q'2}|$. Fix $n = \max_{q'}\,(|c_{q'1}| + |c_{q'2}|)$, and observe that for all σ, $|[\![M']\!]\,(\sigma\sigma_1) - [\![M']\!]\,(\sigma\sigma_2)| \leq n$. For $\sigma = C^{n+1}$, $|f_1\,(\sigma\sigma_1) - f_1\,(\sigma\sigma_2)| > n$, so M' cannot be equivalent to M_1. In general, by a straightforward application of the pigeon-hole principle, we conclude:

Lemma 1. *Let M be an ACRA whose registers are k-separable. Then the register complexity of the implemented function f is at least k.*

Proof. Assume otherwise, so we have a machine M' with only $k-1$ registers and equivalent to M. Let q be that state of M where separation is achieved. For each $v \in U$, there is a suffix $\sigma_v \in \Sigma^*$ and constant c_v so that the suffix summary of σ_v in q is (v, c_v).

For each state q' of the proposed counter-example machine M', and each register $v \in U$ of M, record the suffix summary of σ_v in q' – $\left(v'_{q'v}, c'_{q'v}\right)$, or $\bot$. Define c_p as:

$$c_p = \max\left(\max_{v \in U} |c_v|, \max_{q', v \in U} \left|c'_{q'v}\right|\right).$$

Consider the state of the machine M' after processing some prefix σ_{pre}. For each suffix σ_v, there must be a register v' so that $|[\![M']\!](\sigma_{pre}\sigma_v) - val(v', \sigma_{pre})| \leq c_p$. Since there are only $k-1$ registers in M' and k suffixes σ_i, it must either be the case that for some pair $u, v \in U$, this condition holds offset from the same register v'.

We assumed the condition: for each $c \in \mathbb{N}$, there is a path σ to q so that $|val(u, \sigma) - val(v, \sigma)| \geq c$ (for all distinct $u, v \in V$). Instantiate this condition with $c = 1 + 4c_p$, and let $\sigma_{pre} = \sigma$ be the witness prefix. Let σ_u, σ_v be the pair of suffixes for which the suffix summaries in q' depend on the same v'. Since $|val(u, \sigma_{pre}) - val(v, \sigma_{pre})| \geq c$, it follows that $|f(\sigma_{pre}\sigma_u) - f(\sigma_{pre}\sigma_v)| \geq c - 2c_p \geq 1 + 2c_p$. However, from our closeness condition, it follows that $|[\![M']\!](\sigma_{pre}\sigma_u) - [\![M']\!](\sigma_{pre}\sigma_v)| \leq 2c_p$, leading to a contradiction.

3.2 Non-separability permits register elimination

Intuition Say we are given an ACRA M, and told that its registers are not k-separable. This can be rewritten in the form of an invariant at each state: for each state q, there is a constant c_q so for every collection $U \subseteq V$ with $|U| = k$, and for every string σ with $\delta(q_0, \sigma) = q$, there must exist distinct $u, v \in U$ with $|val(u, \sigma) - val(v, \sigma)| < c$. For example, with 3 registers x, y, z, this invariant would be $\exists c, |x - y| < c \vee |y - z| < c \vee |z - x| < c$. Now, if we know that $|x - y| < c$, then it suffices to explicitly maintain the value of only one register, and the (bounded) difference can be stored in the state.

Consider machines M_4, M_5 in figure 2.2. While M_4 is the intuitive first solution to the problem of implementing f_4, the difference between registers x, y is always bounded. In both states, the non-separability invariant states $|x - y| \leq 1$, or $-1 \leq x - y \leq 1$. We exploit this to construct M_5, which uses just one register u.

Since we need to track these register differences during execution, the invariants must be inductive: if D_q and $D_{q'}$ are the invariants at states q, q', and $q \rightarrow^a q'$ is a transition in the machine, then it must be the case that $D_q \implies \textsc{wp}(D_{q'}, q, a)$. Here $\textsc{wp}$ refers to the standard notion of the weakest precondition from program analysis: the invariant $D_{q'}$ identifies a set of variable valuations. $\textsc{wp}\,(D_{q'}, q, a)$ is exactly that set of variable valuations val so that $(q, val) \to^a (q', val')$ for some $D_{q'}$-satisfying valuation val'.

The standard technique to make a collection of invariants inductive is strengthening: if $D_q \not\Longrightarrow \textsc{wp}\,(D_{q'}, q, a)$, then D_q is replaced with $D_q \wedge \textsc{wp}\,(D_{q'}, q, a)$, and this process is repeated at every pair of states until fixpoint. This procedure is seeded with the invariants asserting non-separability. However, before the result of this back-propagation can be used in our arguments, we must prove that the method terminates – this is the main technical problem solved in this section.

We now sketch a proof of this termination claim for a simpler class of invariants. Consider the class of difference-bound constraints – assertions of the form $C = \bigwedge_{u,v \in V} a_{uv} < u - v < b_{uv}$, where for each u, v, $a_{uv}, b_{uv} \in \mathbb{Z}$ or $a_{uv}, b_{uv} \in \{-\infty, \infty\}$. Observe that C induces an equivalence relation $\equiv_C$ over the registers: $u \equiv_C v$ iff $a_{uv}, b_{uv} \in \mathbb{Z}$. Let C and C' be some pair of constraints so that $C \not\Longrightarrow C'$, so that the assertion $C \wedge C'$ is strictly stronger than C. Either $C \wedge C'$ relates a strictly larger set of variables – $\equiv_C \subsetneq \equiv_{C \wedge C'}$ – or (if $\equiv_C = \equiv_{C \wedge C'}$) for some pair of registers u, v, the bounds $a'_{uv} < u - v < b'_{uv}$ imposed by $C \wedge C'$ are a strict subset of the bounds $a_{uv} < u - v < b_{uv}$ imposed by C. Observe that the first type of strengthening can happen at most $|V|^2$ times, while the second type of strengthening can happen only after a_{uv}, b_{uv} are established for a pair of registers u, v, and can then happen at most $b_{uv} - a_{uv}$ times. Thus the process of repeated invariant strengthening must terminate. However, the statements asserting non-separability are disjunctions of difference-bound constraints. We show that the above insight is sufficient even for this generalization.

The rest of this subsection is devoted to formalizing the intuition presented above.

Difference bound constraints and well-formed invariants

Definition 5. *A* difference bound constraint *is a conjunction of constraints of the form $a < u - v < b$, for $a, b \in \mathbb{Z} \cup \{-\infty, \infty\}$ (and either a, b are both finite, or both infinite), and $u, v \in V$.* Well-formed invariants *are finite disjunctions of difference bound constraints.*

Note that if there is a non-trivial term corresponding to $u - v$ in a difference bound constraint, then the difference is bounded both from above and below, i.e. $a < u - v < b$, and $a, b \in \mathbb{Z}$. For example, $0 < u - v < \infty$ is not a difference bound constraint. The trivial constraint $-\infty < u - v < \infty$ holds of every pair of registers. Given a difference bound constraint C, it can be set in *closed form* where whenever C contains the term $a < u - v < b$ it also contains $-b < v - u < -a$, and if C contains the terms $a < u - v < b$ and $a' < v - w < b'$, then it also contains the term $a'' < u - w < b''$, for some $a + a' \leq a'' \leq b'' \leq b + b'$. A difference bound constraint establishes an equivalence relation over the registers of V, where $u \equiv v$ iff there is a constant c so that $C \Longrightarrow |u - v| < c$. This is the same as saying that $u \equiv v$ iff C in closed form contains a non-trivial term corresponding to $u - v$. The following proposition describes exactly the cases when a difference-bound constraint C is strictly stronger than another constraint C':

Claim. Let $C = c_1 \wedge c_2 \wedge \ldots \wedge c_k$ and $C' = c'_1 \wedge c'_2 \wedge \ldots \wedge c'_{k'}$ be difference bound constraints. If C is strictly stronger than C', i.e. $C \implies C'$ but $C' \not\Longrightarrow C$, then either

1. $\equiv' \subsetneq \equiv$, where $\equiv$, $\equiv'$ are the equivalence relations over V generated by C , C', or
2. (otherwise if $\equiv' = \equiv$) for some registers $u, v \in V$, the best bounds $a < u - v < b$ and $a' < u - v < b'$ implied by C and C' are related as $\{a, a+1, a+2, \ldots, b\} \subsetneq \{a', a'+1, a'+2, \ldots, b'\}$.

Well-formed invariants are well-ordered

Lemma 2. *Let T be a labeled tree, where each node u is labeled with a difference bound constraint C_u, and is of finite degree. Say also that the constraint at each node is strictly stronger than the constraint at its parent. Then T cannot be infinite.*

Proof. Assume otherwise. By König's lemma, there must be an infinite path through this tree, and the constraints along this path strictly increase in strength. We now argue that such a path cannot exist.

Observe that the equivalence relation $\equiv$ associated with a difference bound constraint C can have no more than $|V|^2$ elements. Also, once we have a pair of registers constrained as $a < u - v < b$, (with both a, b finite), the constraint can be tightened only $b - a$ times. Furthermore, such tightening can only happen after $u \equiv v$, by the equivalence relation $\equiv$ associated with C. Thus, every sequence of difference bound constraints strictly increasing in strength must be finite. This completes the proof.

Definition 6. *Let $\varphi(val)$ be an arbitrary formula that identifies sets of states. Let $q, q' \in Q$ be two states so that $q' \rightarrow^a q$ for some symbol $a \in \Sigma$. Then, the weakest precondition of φ at q with respect to the transition from q' on a, written as $\varphi' = \textsc{wp}(\varphi, q', a)$ is $\varphi'(val') \iff \forall val, (q', val') \rightarrow^a (q, val) \implies \varphi(val)$.*

It can be shown that $\textsc{wp}(\varphi, q', a)$ can be obtained by simultaneously replacing every occurrence of each register with its update expression over the transition: $\varphi' = \varphi[v \mapsto \mu(q', a, v)]_v$, where the update expression $\mu(q', a, v) = (u, c)$ is read as "$u + c$". For example, consider machine M_4 in figure 2.2a: the weakest precondition of the assertion $-2 < x - y < 2$ in state q_1 with respect to the transition on b from q_0 is the assertion $-2 < x + 1 - y < 2$, or $-3 < x - y < 1$. It can be shown that:

Claim. 1. Let $D_{q'}$ be a well-formed invariant in some state q' of an ACRA M. Let $q \in Q$ and $a \in \Sigma$ so $\delta(q, a) = q'$. Then $\textsc{wp}(D_{q'}, q, a)$ is also a well-formed invariant.

2. Let D and D' be well-formed invariants. Then so is $D \wedge D'$.

Algorithm 1 SATURATE. Given an ACRA M, and a well-formed invariant D_q at each state $q \in Q$. The algorithm returns an inductive strengthening of these invariants.

```
1. At each state $q$, initialize a tree $T_q$. Nodes of this tree are labeled with difference
   bound constraints. The root of each tree $T_q$ is $true$, and its immediate children are
   the difference bound constraints $C$ in $D_q$.
2. While there exist states $q, q' \in Q$ and symbols $a \in \Sigma$, so that $\delta(q, a) = q'$, but
   $D_q \not\Rightarrow \text{WP}(D_{q'}, q, a)$. For each difference bound constraint $C \in D_q$ so that
   $C \not\Rightarrow \text{WP}(D_{q'}, q, a)$:
   (a) Calculate $C \wedge \text{WP}(D_{q'}, q, a)$, by the distributivity of the logical AND operator
       over OR.
   (b) For the node corresponding to $C$ in $T_q$, create children corresponding to each
       disjunct in $C \wedge \text{WP}(D_{q'}, q, a)$.
   (c) Replace $C$ at $D_q$ with the disjuncts in $C \wedge \text{WP}(D_{q'}, q, a)$.
3. Return, for each state $q$, the well-formed constraint $D_q$.
```

Lemma 3. *For every input $(M, D_{q \in Q})$, algorithm 1 terminates.*

Proof. Observe that with each iteration of the loop in step 2, the size of T_q increases, for some q. If the algorithm were to not terminate, then for some q, T_q would be infinite. We maintain the invariant that each node in T_q has finite degree, and the difference bound constraint at each node is strictly stronger than that at its predecessor. But lemma 2 tells us that no such infinite tree T_q can exist.

Putting it all together: Constructing M'

Lemma 4. *Consider an ACRA M whose registers are not k-separable. Then, we can effectively construct an equivalent machine M' with only $k-1$ registers.*

Proof. The idea is that the difference bounds allow us to track all but $k-1$ registers in the state. So some registers u are represented in the state as a pair (v, c), and we simulate the effect of register u by the expression $v + c$.

Since the registers of M are not k-separable, at each state q, and collection of k registers U, there is a constant c so for all paths σ going to q, there is some pair of distinct registers $u, v \in U$ so that $|val(u, \sigma) - val(v, \sigma)| < c$ (or equivalently, $-c < u - v < c$). Since $U \in 2^V$ is drawn from a finite set, and any instantiation of c can be replaced by a larger constant $c' \geq c$, we can change the order of quantifiers: at each state q, there is a constant c, so for all paths σ going to q and collections of k registers $U \subseteq V$, there exist distinct $u, v \in U$ so that $|val(u, \sigma) - val(v, \sigma)| < c$. Simplifying this, we obtain at each state q, a well-formed invariant D_q. In each disjunct C in D_q, there is never a collection of more than $k-1$ mutually unrelated registers. Run SATURATE on these constraints to make them inductive.

Now construct M' as follows. Consider some state q and some difference bound constraint $C \in D_q$. Now arbitrarily pick a maximal set $V_{q,C} \subsetneq V$ of registers so no two elements $u, v \in V_{q,C}$ are constrained by C. Since this set is maximal, for every register $u \in V \setminus V_{q,C}$, there is a register $v \in V_{q,C}$ so we have $C \implies a_{q,C,u} \leq u - v \leq b_{q,C,u}$, for $a_{q,C,u}, b_{q,C,u} \in \mathbb{Z}$. Define the state space Q' of M' as:

$$Q' = \bigcup_{q,C \in D_q} \left(\{(q, C)\} \times \prod_{u \in V \setminus V_{q,C}} [a_{q,C,u}, b_{q,C,u}] \right),$$

where $[a_{q,C,u}, b_{q,C,u}]$ is the set of integers $a_{q,C,u} \leq z \leq b_{q,C,u}$. Thus, for example, if we have 3 registers x, y, z, and at state q, we have the invariant that $-2 \leq x - y \leq 3$, and $0 \leq z \leq 1$, then q would produce states $\{(q, -2, 0), (q, -2, 1), (q, -1, 0), (q, -1, 1), (q, 0, 0), (q, 0, 1), \ldots, (q, 3, 1)\}$. Also, $V_{q,C}$ never has more than $k - 1$ registers.

Now define $\delta' : Q' \times \Sigma \to Q'$. Let $(q, C, \mathbf{v}) \in Q'$ be a state, where $\mathbf{v}$ refers to the values of the offsets. Let a be a symbol, and let $\delta(q, a) = q'$. Since the invariants are inductive, it follows that there is a difference bound constraint C' at q' which holds when the machine makes this transition with this precondition. Also, there is enough information to determine statically the values of the offsets $\mathbf{v}'$. Define $\delta'((q, C, \mathbf{v}), a) = (q', C', \mathbf{v}')$.

Let $k' = \max_{q,C} |V_{q,C}|$. Define V' to have k' registers. At each state-constraint pair q, C, choose an arbitrary mapping scheme which maps registers $v' \in V'$ to registers $v \in V_{q,C}$. The invariant is that for all paths to $(q, C, \mathbf{v})$, v' holds the value of the corresponding register v. For every register $u \in V \setminus V_{q,C}$, the offsets in $\mathbf{v}$ provide enough information to simulate its value by the expression $v + c$. Because the invariants are inductive, there is enough local information to define the register update function μ', and the output function ν'.

The start state q' is any triple $(q_0, C, \mathbf{0})$, where C is any constraint at q_0 satisfied initially. All registers start at 0, so all register differences start at 0 also. Observe that the machine M' is equivalent to M by construction, and has $k' < k$ registers. This completes the proof.

It should be noted that there is considerable freedom when defining the reduced machine M' above: the start state $(q_0, C, \mathbf{0})$ is not necessarily unique – any difference-bound constraint $C \in D_{q_0}$ which is initially satisfied will work. Also, there may be multiple difference-bound constraints C'_1, C'_2, ..., that are satisfied at q' when making a transition on symbol a from $(q, C, \mathbf{x})$. The choice in such cases can be made arbitrarily.

Example 1. Consider machine M_3 in figure 2.1b. By construction, we know that register x always holds the same value as one of the registers y, z. In particular, we have $|x - y| \leq 0 \vee |y - z| \leq 0 \vee |z - x| \leq 0$ as the non-separation invariant. The weakest precondition with respect to the transition from q on a is $|(y + 1) - (y + 1)| \leq 0 \vee |(y + 1) - z| \leq 0 \vee |z - (y + 1)| \leq 0$, which is always true. Thus, $D_q \implies \mathrm{WP}(D_q, q, a)$, and similarly $D_q \implies \mathrm{WP}(D_q, q, b)$. Algorithm 1 returns immediately. We then construct the 3 state machine shown in figure 3.1. State q_{xy} encodes the triple $(q_0, x = y, 0)$, and similarly for q_{yz} and q_{zx}. The machine maintains 2 registers u, v. The state-specific mapping of these to the original registers are: in q_{xy}, u, v hold x, z, in q_{yz}, u, v hold x, y, and in q_{zx}, u, v hold z, y respectively. Any of the states could be marked as the start state.

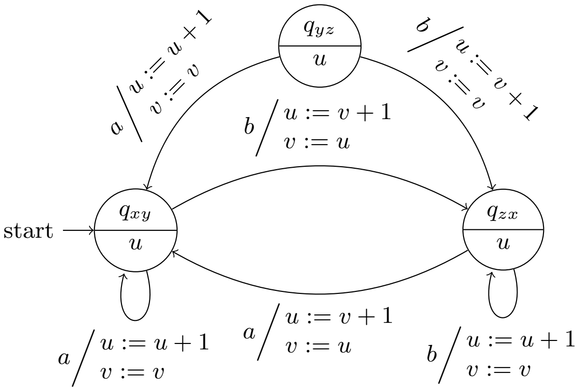

q_{yz}
u
q_{xy}
u
q_{zx}
u
start
a / $u := u + 1$, $v := v$
b / $u := v + 1$, $v := v$
b / $u := v + 1$, $v := u$
a / $u := u + 1$, $v := v$
a / $u := v + 1$, $v := u$
b / $u := u + 1$, $v := v$

Fig. 3.1: An example application of lemma 4 to M_3.

4 Computing the Register Complexity

4.1 Computing the register complexity is in PSPACE

Intuition We reduce the problem of determining the register complexity of $[\![M]\!]$ to one of determining reachability in a directed "register separation" graph with $O\left(|Q|\, 2^{|V|^2}\right)$ nodes. The presence of an edge in this graph can be determined in polynomial space, and thus we have a PSPACE algorithm to determine the register complexity. Otherwise, if polynomial time algorithms are used for graph reachability and 1-counter 0-reachability, the procedure runs in time $O\left(c^3\, |Q|^4\, 2^{4|V|^2}\right)$, where c is the largest constant in the machine.

We first generalize the idea of register separation to that of separation relations: an arbitrary relation $\parallel \subseteq V \times V$ separates a state q if for every $c \in \mathbb{Z}$, there is a string σ so that $\delta(q_0, \sigma) = q$, and whenever $u \parallel v$, $|val(u, \sigma) - val(v, \sigma)| \geq c$. Thus, the registers of M are k-separable iff for some state q and some subset U of live registers at q, $|U| = k$ and $\{(u, v) \mid u, v \in U, u \neq v\}$ separates q.

Consider a string $\tau \in \Sigma^*$, so for some q, $\delta(q, \tau) = q$. Assume also that:

1. For every register u in the domain or range of $\parallel$, $\mu(q, \tau, u) = (u, c_u)$, for some $c_u \in \mathbb{Z}$, and
2. for some pair of registers x, y, $\mu(q, \tau, x) = (x, c)$ and $\mu(q, \tau, y) = (y, c')$ for distinct c, c'.

Thus, every pair of registers that is already separated is preserved during the cycle, and some new pair of registers is incremented differently. We call such strings τ "separation cycles" at q. They allow us to make conclusions of the form: If $\|$ separates q, then $\| \cup \{(x, y)\}$ also separates q.

Now consider a string $\sigma \in \Sigma^*$, so for some q, q', $\delta(q, \sigma) = q'$. Pick arbitrary relations $\|$, $\|'$, and assume that whenever $u' \;\|'\; v'$, and $\mu(q, \sigma, u') = (u, c_u)$, $\mu(q, \sigma, v') = (v, c_v)$, we have $u \parallel v$. We can then conclude that if $\|$ separates q, then $\|'$ separates q' We call such strings σ "renaming edges" from $(q, \|)$ to $(q', \|')$.

We then show that if $\|$ separates q and $\|$ is non-empty, then there is a separation cycle-renaming edge sequence to $(q, \|)$ from some strictly smaller separation $(q', \|')$. Thus, separation at each node can be demonstrated by a sequence of separation cycles with renaming edges in between, and thus we reduce the problem to that of determining reachability in an exponentially large register separation graph. Finally, we show that each type of edge can be determined in PSPACE.

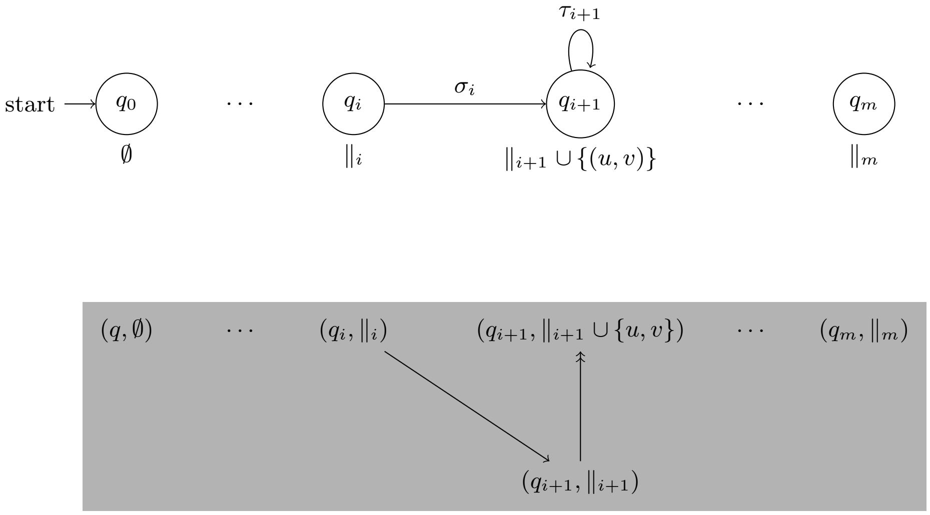


Fig. 4.1: The register separation graph. String σ_i "renames" the separation $\|_i$ into $\|_{i+1}$, and cycle τ_{i+1} creates a separation between u and v, while preserving all previously created separations. The goal is to reach a separation $\|_m$ which has a k-clique of live registers.

Register separation graphs

Definition 7. *Consider some ACRA M, and let $\| \in 2^{V \times V}$ be a relation over V. We say that $\|$ separates q if for every constant $c \in \mathbb{N}$, there exists a string σ so $\delta(q_0, \sigma) = q$ and for all $u, v \in V$, if $u \parallel v$, then $|val(u, \sigma) - val(v, \sigma)| \geq c$. Also,* *we say that a string σ c-separates $(q, \|)$, if $\delta(q_0, \sigma) = q$, and for every $(u, v) \in \|$, $|val(u, \sigma) - val(v, \sigma)| \geq c$.*

Definition 8. *Consider the set $2^{V \times V}$ of relations over V. The* register separation graph *has nodes $Q \times 2^{V \times V} \cup \{t\}$, and the following edges (figure 4.1):*

1. *(Separation edges). From $(q, \|)$ to $(q, \| \cup \{(u, v)\})$ if there is a cycle σ at q so that $\mu(q, \sigma, u) = (u, c)$, $\mu(q, \sigma, v) = (v, c')$, $c \neq c'$, and for each w in the domain or range of $\|$, $\mu(q, \sigma, w) = (w, c_w)$, for appropriate $c_w \in \mathbb{Z}$.*
2. *(Renaming edges). From $(q, \|)$ to $(q', \|')$ if for some string σ that leads q to q', whenever $(u, v) \in \|'$, $\mu(q, \sigma, u) = (u', c)$ and $\mu(q, \sigma, v) = (v', c')$, and $u' \| v'$.*
3. *(Final edges). From $(q, \|)$ to t, if there is a collection $U \subseteq V$ of k registers, $|U| = k$, so for each distinct pair $u, v \in U$, $u \| v$.*

Informally, a separation edge identifies a cycle τ which increments a pair of registers u, v differently, while all other relevant registers flow into themselves. Renaming edges effect a "renaming" of the separation $\|$ at q into a separation $\|'$ at q'. Final edges to the sink node t exist simply to identify a uniform target vertex. They are triggered only from vertices where k-separation has already been achieved.

The algorithm is to find a path through the register separation graph from $(q_0, \emptyset)$ to t. We first show that a path exists in the register separation graph from $(q_0, \emptyset)$ to $(q, \|)$ iff $\|$ separates q. But since the presence of a single edge in this graph can be determined in polynomial space, and the "current node" can be stored in $O\left(|V|^2 \log |Q|\right)$ space, the presence of such a path can also be determined in polynomial space. Lemmas 5, 6, and 7 are the three steps to show the correctness of this approach.

Connecting k-separability to register separation graphs

Lemma 5. *If there is a path π from $(q_0, \emptyset)$ to $(q, \|)$ in the register separation graph, then $\|$ separates q.*

Proof. Informally, since every register pair $(u, v) \in \|$ are separated by some separation edge in π, and no subsequent edge results in the resetting of this difference (though they might increase or decrease the difference), the cycle can be passed enough times to create a sufficiently large separation.

Say there are m separation edges in π. Then by definition, for every vector $\mathbf{x} \in \mathbb{N}^m$, there is a string σ to q so that for all $u \| v$,

$$val(u, \sigma) - val(v, \sigma) = c_{uv} + \sum_i d_i^{uv} x_i,$$

where d_i^{uv} is the difference created between u and v by the i^{th} separation edge in π. Also, by construction, for each $u \| v$, there is an i so that $d_i^{uv} \neq 0$.

If we construct a vector $\mathbf{x}$ so that $\sum_i d_i^{uv} x_i$ are simultaneously non-zero for all u, v, we are done, for then by appropriately scaling $\mathbf{x}$, $val\,(u, \sigma) - val\,(v, \sigma)$ can be made arbitrarily large in magnitude. Choose $x_1 = 1$, and once $x_1, \ldots, x_i$ are defined, define

$$x_{i+1} = 1 + \max_{u \| v, d_{i+1}^{uv} \neq 0} \left\lceil \frac{\sum_{j \leq i} \left| d_j^{uv} \right| x_j}{\left| d_{i+1}^{uv} \right|} \right\rceil . \tag{4.1}$$

(In the degenerate case when $d_{i+1}^{uv} = 0$ for all u, v, choose an arbitrary value for x_{i+1}) This has the property that $\left| d_{i+1}^{uv} x_{i+1} \right| > \sum_{j \leq i} d_j^{uv} x_j$ (if d_{i+1}^{uv} is non-zero), and so $\sum_i d_i^{uv} x_i$ is non-zero for all $u \parallel v$. This completes the proof.

Lemma 6. *If $\|$ separates q, then there is a path through the register separation graph from $(q_0, \emptyset)$ to $(q, \|)$.*

Proof. Consider some pair $(u, v) \in \|$ – since u and v are separable at q, intuitively it has to be the case that there is a cycle τ resulting in different increments to u and v (or a path from some other state q' where u' and v' were differently incremented on τ', and then the values of these registers flowed into u and v respectively). We now formalize this intuition

By induction on the number of elements in $\|$. There is a path from $(q_0, \emptyset)$ to $(q, \emptyset)$, for every reachable state q. Say $\|$ has $m + 1$ elements, and the proposition holds at every q for every $\|'$ with at most m elements each. We now show the existence of a reachable vertex $(q_l, \|_l)$, where $\|_l$ has m elements, and there is a path from $(q_l, \|_l)$ to $(q, \|)$.

Consider some state q', which on reading symbol a transitions to q. We define the weakest precondition of $\|$ with respect to this transition as the smallest relation $\|' \subseteq V \times V$ so that whenever $u \parallel v$ then $u' \parallel' v'$, where $\mu\,(q', a, u) = (u', c_u)$ and $\mu\,(q', a, v) = (v', c_v)$. Observe that whenever $\|$ separates q, there must be a predecessor state q' transitioning to q on some symbol a so that the weakest precondition of $\|$ with respect to this transition, $\|'$ separates q' (for otherwise, along every path to q, because of the unreachability of the predecessor separation, some registers $u \parallel v$ have to be close).

Specifically, let $N \subseteq Q \times 2^{V \times V}$ be a set of vertices in the register separation graph. Then, for sufficiently large c, there is a constant c' and an $N' \subseteq Q \times 2^{V \times V}$ of weakest precondition separations so that all strings σ that c-separate some element of N must be at least one symbol long, and $\sigma_1 \ldots \sigma_{|\sigma|-1}$ must c'-separate some element of N'. If we start with $N = \{(q, \|)\}$, and repeat this $n = (p+1)\,2^p$ times (where p is the number of vertices in the register separation graph), then some subset N' must be repeated at least $p + 1$ times, let these positions be i_1, $\ldots$, i_{p+1}, indexed from the end. Let c_n be the separation at $N = \{(q, \|)\}$ so this process can be repeated n times. Choose the shortest string σ that c_n-separates $(q, \|)$. (Indexing σ from the end) At least two of $\sigma_{i_1}, \ldots, \sigma_{i_{p+1}}$ must pass through the same state q_l, and separate the same subset of registers $\|_l'$. Let the cycle between these occurrences be τ, so $\sigma = \sigma' \tau \sigma''$, and $\tau \neq \epsilon$. For each pair $(u, v) \in \|_l'$, consider the register separations after processing σ' and $\sigma' \tau$. If no difference changes, then $\sigma'\sigma''$ also c_n-separates $(q, \|)$, contradicting the assumption that σ was the shortest such string. Thus, some pair of registers $(u, v) \in \|'_l$, must have been incremented differently through this cycle. Define $\|_{\mathrm{l}} = \|'_{\mathrm{l}} \setminus \{(u, v)\}$, so that both edges $(q_l, \|_l) \to (q_l, \|'_l) \to (q, \|)$ are present in the register separation graph. $\|_l$ separates q_l, and possesses only m elements. Hence the proof.

Putting it all together

Lemma 7. *Let $(q, \|)$ and $(q', \|')$ be nodes in the register separation graph. The problem of determining whether an edge exists between $(q, \|)$ and $(q', \|')$ can be answered in polynomial space.*

Proof. An edge between two nodes in the register separation graph is either a cycle edge or a renaming edge. We treat the three cases separately:

1. Whether a renaming edge exists between $(q, \|)$ and $(q', \|')$ can be done in non-deterministic polynomial space. We simply guess the witness string $\sigma \in \Sigma^*$ from q to q', one symbol at a time, and update the current register q_t and separation $\|_t$. We accept if $q_t = q'$ and $\|' \subseteq \|_{\mathrm{t}}$. This is essentially a graph-reachability query which is solvable in $O\left(\log |Q| \, 2^{|V|^2}\right)$ non-deterministic space.
2. To determine the presence of a cycle edge, we first observe that it is an instance of a 1-counter non-zero reachability problem. A 1-counter machine is a tuple $A = (Q_A, \delta, q_0)$, where $\delta \subseteq Q_A \times Q_A \times \mathbb{Z}$, and $q_0 \in Q_A$. The semantics are non-deterministic: we start in state q_0, with the counter initialized to 0. If we are currently in a state $q \in Q_A$, then we can transition to any state q' so that $(q, q', c) \in \delta$. During this transition, the counter is incremented by c. Given a final state $q \in Q_A$, the non-zero reachability problem asks: is there a path from q_0 to q so that the counter value is non-zero? In our case, the counter encodes the difference between two registers u' and v', whose values have been influenced by the initial values of u and v respectively. The states $(q, f) \in Q_A$ encode the current state $q \in Q$, and the current register renaming $f : V \to V$, i.e. for each register v, $f(v)$ tells us the name of the initial register whose value has flowed into v. Observe that Q_A is large: it has $O\left(|Q| \, |V|^{|V|}\right)$ states, and thus we never explicitly construct A. We recall from [1] that the 1-counter 0-reachability problem is in NLOGSPACE, and can be answered in $O(\log c \, |Q_A|)$ non-deterministic space, where c is the largest constant appearing in the definition of A. From this, it follows that the non-zero reachability problem can also be solved in $O(\log c \, |Q_A|)$ non-deterministic space. Thus, the presence of a cycle edge can be determined in $O\left(\log c \, |Q| \, |V|^{|V|}\right) = O(\log c \, |Q| + |V| \log |V|)$ non-deterministic space.
3. To determine the presence of a final edge from $(q, \|)$ to t, we simply guess the k-clique U of separated registers. This can be done in $O(|V|)$ non-deterministic space.

We now have the main result of this section:

Theorem 2. *Given an ACRA M and a number k, there is a* PSPACE *procedure to determine whether its register complexity is at least k.*

Proof. We know that the registers of M are k-separable iff there is a path through the register separation graph from $(q_0, \emptyset)$ to t.

Observe that the register separation graph has $O\left(|Q|\, 2^{|V|^2}\right)$ nodes. Since graph reachability can be solved in NLOGSPACE, this problem can be solved in $O\left(\log |Q| + |V|^2\right)$ non-deterministic space. Putting the procedures together – separating loop detection requires $O\left(\log c\,|Q| + |V| \log |V|\right)$, renaming edge detection needs $O\left(\log |Q| + |V|^2\right)$, and final edge detection needs $O\left(|V|\right)$ non-deterministic space. It follows that the register complexity can be determined using $O\left(\log c\,|Q| + |V|^2\right)$ non-deterministic space.

An alternative in the above procedure is to use fast polynomial time algorithms as subroutines: Reachability in a graph with n vertices can be determined in $O(n)$ time, and 1-counter 0-reachability of an n state machine can be decided in $O\left((cn)^3\right)$ time. With this assumption, the procedure runs in $O\left(n\left(n + (cn)^3 + 2^{|V|}\,|V|^2\right)\right)$ time with $n = |Q|\, 2^{|V|^2}$, and c is the largest constant in M, giving the final time complexity of the algorithm as $O\left(c^3\,|Q|^4\, 2^{4|V|^2}\right)$.

4.2 Pumping lemma for ACRAs

The following theorem is the interpretation of a path through the register separation graph. Given a regular function f of register complexity at least k, it guarantees the existence of m cycles τ_1, ..., τ_m, serially connected by strings σ_0, ..., σ_m, so that based on one of k suffixes w_1, ..., w_k, the cost paid on one of the cycles must differ. These cycles are actually the separation cycles discussed earlier, and intermediate strings σ_i correspond to the renaming edges. Consider for example, the function f_2 from figure 2.1, and let $\sigma_0 = \epsilon$, $\tau_1 = aab$, and $\sigma_1 = \epsilon$. We can increase the difference between the registers x and y to arbitrary amounts by pumping cycle τ_1. Now if the suffixes are $w_1 = a$, and $w_2 = b$, then the choice of suffix determines the "cost" paid on each iteration of the cycle.

Theorem 3. *A regular function $f : \Sigma^* \to \mathbb{Z}_\perp$ has register complexity at least k iff there exist strings σ_0, ..., σ_m, τ_1, ..., τ_m, and suffixes w_1, ..., w_k, and k distinct coefficient vectors $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{Z}^m$ so that for all vectors $\mathbf{x} \in \mathbb{N}^m$,*

$$f\left(\sigma_0 \tau_1^{x_1} \sigma_1 \tau_2^{x_2} \ldots \sigma_m w_i\right) = \sum_j c_{ij} x_j + d_i.$$

Proof. We deal with the two cases separately:

1. If f has register complexity at least k, then there is a path π through the register separation graph to a vertex $(q, \|)$ with a k-clique of live registers

in $\|$. Every such path can be collapsed into one where this is exactly one renaming edge (possibly corresponding to ϵ) between any two cycle edges. Let σ_i be the $(i+1)^{\text{th}}$ renaming edge, and let τ_i be the i^{th} cycle edge. Since k mutually divergent registers are live, for each such register v, there exists a suffix w_v to extract its value. By the definition of the register separation graph, the claim follows.

2. Say there exist strings $\sigma_0, \ldots, \sigma_m, \tau_1, \ldots, \tau_m, w_1, \ldots, w_k$ so that this holds. Since there are only finitely many states in any given machine M implementing f, there must exist i_1, j_1 so that $\delta\left(q_0, \sigma_0 \tau_1^{i_1}\right) = \delta\left(q_0, \sigma_0 \tau_1^{i_1} \tau_1^{j_1}\right) = q_1$, for some $q_1 \in Q$. Similarly, there must be i_2, j_2 so that $\delta\left(q_1, \sigma_1 \tau_2^{i_2}\right) = \delta\left(q_1, \sigma_1 \tau_2^{i_2} \tau_2^{j_2}\right) = q_2$, for appropriate q_2. Repeat this process to reach state q_{m+1}. It now follows that there must exist at least k separable registers in q_{m+1}, since a divergent value is extracted by each w_i. Thus, the register complexity of f is at least k.

4.3 Computing the register complexity is PSPACE-hard

We reduce the DFA intersection non-emptiness checking problem to the problem of computing the register complexity. Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be a DFA. Consider a single-state ACRA M with input alphabet Σ. For each state $q \in Q$, M maintains a register v_q. On reading a symbol $a \in \Sigma$, M updates $v_q := v_{\delta(q,a)}$, for each q. Observe that this is simulating the DFA in reverse: if we start with a special tagged value in v_{q_f}, then after processing σ, that tag is in v_{q_0} iff σ^{rev} is accepted by A. Also observe that doing this in parallel for all the DFAs no longer requires an exponential product construction, but only as many registers as a linear function of the input size. We use this idea to construct in polynomial time an ACRA M whose registers are $(k+2)$-separable iff there is a string $\sigma \in \Sigma^*$ which is simultaneously accepted by all the DFAs.

Lemma 8. *The following problem is* PSPACE*-complete [10]: Given a set of DFAs, $\mathcal{A} = \{A_1, \ldots, A_k\}$ over a common input alphabet Σ, is the intersection of their languages non-empty?*

In particular, the problem remains hard if we restrict the DFAs to have a single accepting state each, for a DFA over any alphabet could be extended with a new end-of-string symbol, and made to possess a single accepting state (incurring only a constant size increase).

Claim. The following problem is PSPACE-complete: Given a set of DFAs, $\mathcal{A} = \{A_1, \ldots, A_k\}$ over a common input alphabet Σ, and each with a single accepting state, is the intersection of their languages non-empty?

In figure 4.2, we describe the reduction informally. Unlabelled transitions are triggered by special control symbols not in Σ. For each state q of each DFA A_i, the ACRA maintains a register v_q. Consider the self-loop in state q_1 of the separation gadget: on reading symbol $a \in \Sigma$, each register v_q is assigned the value of $v_{\delta(q,a)}$. Thus, after reading a string $\sigma \in \Sigma^*$, v_q contains the value initially in $v_{\delta(q,\sigma^{rev})}$, where σ^{rev} is the reverse string of σ. The initial loop at q_0 sets up large distinct values in all the final states. Thus, any string σ that is simultaneously accepted by all DFAs corresponds to a way of reaching q_f with large values in $v_{q_{0i}}$, the registers corresponding to the initial states. The self-loop at q_f sets up a large value in a special register u. Therefore, if the DFAs accept a common string, then q_f is $(k+2)$-separable. If no string is accepted by all DFAs, then on each path to q_f, $v_{q_{0i}} = 0$, for some i, and hence q_f is not $(k+2)$-separable. Furthermore, along each path to q_0 or q_1, all registers contain one of at most $k+1$ distinct values, and there is exactly one live register in each q_{outi}. Therefore no state other than q_f is $(k+2)$-separable. Thus, the registers of the separation gadget are $(k+1)$-separable iff all the DFAs simultaneously accept some string.

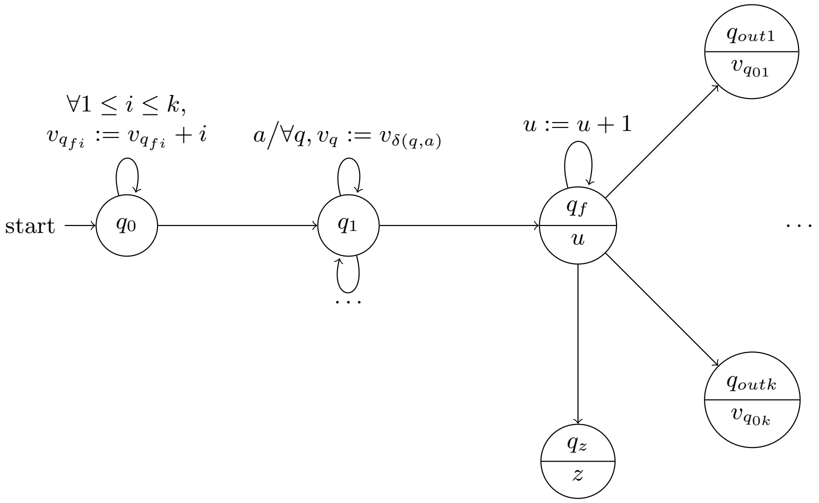


Fig. 4.2: The separation gadget. In the self-loop at q_1, δ refers to the transition function of the appropriate DFA.

Definition 9. *Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be a set of k DFAs, each with a single accepting state. The* separation gadget *of $\mathcal{A}$ is the following ACRA $M = (Q, \Sigma', V, \delta, \mu, q_0, \nu)$:*

1. $Q = \{q_0, q_1, q_f, q_z\} \cup \{q_{outi} \mid 1 \le i \le k\}$,
2. $\Sigma' = \Sigma \cup \{\#\} \cup \{a_i \mid 1 \le i \le k\}$, and
3. $V = \{u, z\} \cup \{v_q \mid q \in Q_i, 1 \le i \le k\}$.
4. δ is defined by the following rules:
   (a) $\delta(q_0, \#) = q_1$. For all other $a \in \Sigma'$, $\delta(q_0, a) = q_0$.
   (b) For each $a \in \Sigma$, $\delta(q_1, a) = q_1$. For all other $a \in \Sigma'$, $\delta(q_1, a) = q_f$.
   (c) For all $a \in \Sigma$, $\delta(q_f, a) = q_f$. $\delta(q_f, \#) = q_z$. For each a_i, $1 \le i \le k$, $\delta(q_f, a_i) = q_{outi}$.

(d) For each i, $1 \leq i \leq k$, and $a \in \Sigma'$, $\delta\left(q_{outi}, a\right) = q_{outi}$.

5. μ is defined by the following rules:

   (a) For all $a \in \Sigma'$ so $\delta\left(q_0, a\right) = q_0$, and $1 \leq i \leq k$, $\mu\left(q_0, a, v_{q_{fi}}\right) = \left(v_{q_{fi}}, i\right)$.
   (b) For all q, $a \in \Sigma$, $\mu\left(q_1, a, v_q\right) = \left(v_{\delta'(q,a)}, 0\right)$. Here δ' is the transition function of the DFA containing q'.
   (c) For all $a \in \Sigma$, $\mu\left(q_f, a, u\right) = (u, 1)$.
   (d) For all other q, a, v, $\mu\left(q, a, v\right) = (v, 0)$.

6. $\nu\left(q_f\right) = (u, 0)$, $\nu\left(q_z\right) = (z, 0)$, and $\nu\left(q_{outi}\right) = \left(v_{q_{0i}}, 0\right)$, for all i. In all other states, $\nu\left(q\right) = \bot$.

Proposition 1. *Let $\mathcal{A}$ be a set of k DFAs, and M be the separation gadget of $\mathcal{A}$.*

1. *Let $\sigma \in \left(\Sigma'\right)^*$ so $\delta\left(q_0, \sigma\right) \notin \left\{q_f, q_z, q_{outi}\right\}$. Then there is a collection $P \subseteq \mathbb{Z}$ with $|P| \leq k + 1$, so for each register $v \in V$, $val\left(v, \sigma\right) \in P$.*
2. *If the intersection language of the DFAs is empty, then for each $\sigma \in \left(\Sigma'\right)^*$, if $\delta\left(q_0, \sigma\right) = q_f$, there is some i so that $val\left(q_{0i}, \sigma\right) = 0 = val\left(z, \sigma\right)$.*
3. *If the intersection language of the DFAs is non-empty, then for each $c \in \mathbb{Z}$, there is a $\sigma \in \left(\Sigma'\right)^*$ so that $\delta\left(q_0, \sigma\right) = q_f$, and for each $v, v' \in \{u, z\} \cup \left\{q_{0i} \mid 1 \leq i \leq k\right\}$, $\left|val\left(v, \sigma\right) - val\left(v', \sigma\right)\right| \geq c$.*

We now conclude the hardness argument:

Theorem 4. *Given an ACRA M and a number k, deciding whether the register complexity of $[\![M]\!]$ is at least k is* PSPACE*-hard.*

Proof. Given a set of k DFAs $\mathcal{A}$, the separation gadget M of $\mathcal{A}$ can be constructed in polynomial time (M has $k + 3$ states, $2 + \sum_i |Q_i|$ registers, and operates over an alphabet of $k + |\Sigma| + 1$ symbols). From proposition 1, it follows that an equivalent ACRA with $k + 1$ registers exists iff the intersection language is empty. Thus, the problem is PSPACE-hard.

5 Games over ACRAs

We now study games played over ACRAs. We extend the model of ACRAs to allow alternation – in each state, a particular input symbol may be associated with multiple transitions. The system picks the input symbol to process, while the environment picks the specific transition associated with this input symbol. Accepting states are associated with output functions, and the system may choose to end the game in any accepting state. Given a budget k, we wish to decide whether the system has a winning strategy with worst-case cost no more than k. We show that ACRA games are undecidable when the registers are integer-valued, and EXPTIME-complete when the domain is $\mathbb{D} = \mathbb{N}$.

Definition 10. *An ACRA $(\mathbb{D})$ reachability game is played over a structure $G = (Q, \Sigma, V, \delta, \mu, q_0, F, \nu)$, where Q, Σ, and V are finite non-empty sets of states, input symbols and registers respectively, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $\mu : \delta \times V \to V \times \mathbb{D}$ is the register update function, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states, and $\nu : F \to V \times \mathbb{D}$ is the output function.*

The game configuration is a tuple $\gamma = (q, val)$, where $q \in Q$ is the current state, and $val : V \to \mathbb{D}$ is the current register valuation. A run π is a (possibly infinite) sequence of game configurations $(q_1, val_1) \to^{a_1} (q_2, val_2) \to^{a_2} \cdots$ with the property that

1. *the transition $q_i \to^{a_i} q_{i+1} \in \delta$ for each i, and*
2. *$val_{i+1}(u) = val_i(v) + c$, where $\mu(q_i \to^{a_i} q_{i+1}, u) = (v, c)$, for each register u and transition i.*

A strategy is a function $\theta : Q^* \times Q \to \Sigma$ that maps a finite history $q_1 q_2 \ldots q_n$ to the next symbol $\theta(q_1 q_2 \ldots q_n)$. A run π is consistent with θ if for each i, $\theta(q_1 q_2 \ldots q_i) = a_i$. θ is winning starting from a state q if for every run π consistent with θ and starting from $q_1 = q$, there is some i so that $q_i \in F$. It is winning from a configuration (q, val) with a budget of $k \in \mathbb{D}$ if for every consistent run π starting from $(q_1, val_1) = (q, val)$, for some i, $q_i \in F$ and $\nu(q_i, val_i) \leq k$.

For greater readability, we write tuples $(q, a, q') \in \delta$ as $q \to^a q'$. If $q \in F$, and val is a register valuation, we write $\nu(q, val)$ for the result $val(v) + c$, where $\nu(q) = (v, c)$. When we omit the starting configuration for winning strategies it is understood to mean the initial configuration (q_0, val_0) of the ACRA.

Consider the natural partial order $\preceq$ over register valuations: $val \preceq val'$ iff for all registers v, $val(v) \leq val'(v)$. Then, any winning strategy for large valuations is also a winning strategy for small valuations:

Claim. For each q, k, val, val', if $val \preceq val'$, then every strategy θ which is k-winning starting from (q, val') is also k-winning starting from (q, val).

5.1 ACRA $(\mathbb{N})$ reachability games can be solved in EXPTIME

Consider the simpler class of (unweighted) graph reachability games. These are played over a structure $G^f = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite state space, and Σ is the input alphabet. $\delta \subseteq Q \times \Sigma \times Q$ is the state transition relation, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accepting states. If the input symbol $a \in \Sigma$ is played in a state q, then the play may adversarially proceed to any state q' so that $(q, a, q') \in \delta$. The system can force a win if every run compatible with some strategy $\theta^f : Q^* \times Q \to \Sigma$ eventually reaches a state $q_f \in F$. Such games can be solved by a recursive back-propagation algorithm – corresponding to model checking the formula $\mu X \cdot \left(F \vee \bigvee_{a \in \Sigma} [a] X\right)$ – in time $O(|Q| |\Sigma|)$. Observe that these games obey the "small strategy" property: if there is a winning strategy θ, then there is a winning strategy θ_{small} which guarantees that no state is visited twice.

From every ACRA $(\mathbb{N})$ reachability game $G = (Q, \Sigma, V, \delta, \mu, q_0, F, \nu)$, we can project out an unweighted graph reachability game $G^f = (Q, \Sigma, \delta, q_0, F)$. Also,

G^f has a winning strategy iff for some $k \in \mathbb{N}$, G has a k-winning strategy. Consider the cost of θ_{small} (computed for G^f) when used with G. Since no run ever visits the same state twice, θ_{small} is $c_0 \, |Q|$-winning, where c_0 is the largest constant appearing in G. We have thus established an upper-bound on the optimal reachability strategy, if it exists.

Now assume that we are given an upper-bound k, and asked to determine whether a winning strategy θ exists within this budget. Because the register increments are non-negative, once a register v achieves a value larger than k, it cannot contribute to the final output, on any suffix σ permitted by the winning strategy. We thus convert G into an unweighted graph reachability G^f_k, where the value of each register is explicitly tracked in the state, until it is larger than k. After this, its value is clamped down to $k{+}1$.

Definition 11. *Let $G = (Q, \Sigma, V, \delta, \mu, q_0, F, \nu)$ be an ACRA $(\mathbb{N})$ reachability game. Then, for $k \in \mathbb{N}$, define the corresponding graph reachability game*

$$G^f_k = \left(Q' = Q \times [k+1]^{|V|}, \Sigma, \delta', (q_0, \mathbf{0}), F \right)$$

as follows. Here $[k+1] = \{0, 1, 2, \ldots, k+1\}$. Consider some state $(q, \mathbf{x}) \in Q'$, and $a \in \Sigma$. Define $val_{\mathbf{x}} : V \to \mathbb{N}$ as $val_{\mathbf{x}}(u) = x_u$. Say also that $(q, val) \to^a (q', val')$, for some q', val' is a valid transition of the game configuration of G on playing symbol a. Define $\mathbf{x}'_{val}$ as $x'_{val,u} = val'(u)$, if $val'(u) \leq k$. Otherwise $x'_{val,u} = k+1$. Then $((q, \mathbf{x}) \to^a (q', \mathbf{x}'_{val})) \in \delta'$. Define $(q, \mathbf{x}) \in F \iff \nu(q, val_{\mathbf{x}}) \leq k$.

We claim that G has a k-winning strategy θ iff the player can force a win in G^f_k. Consider any state $(q, \mathbf{x}) \in Q'$ from which the player can force a win. By induction on the assertion that $(q, \mathbf{x})$ is winning, we can show there is a k-winning strategy from every configuration (q, val) in G where $\mathbf{x} = \mathbf{x}_{val}$. Conversely, pick a configuration (q, val) of G from which a k-winning strategy θ exists. It follows that θ is also a winning strategy in G^f_k.

Furthermore, the decision procedure for this problem can be translated into an optimization procedure: given an upper bound on the budget k, determine the smallest $k' \leq k$, if exists, so that G has a k'-winning strategy. From our discussion in the main paper, we know that if a winning strategy exists, then there is a winning strategy θ with budget at most $c_0 \, |Q|$. Hence we have:

Theorem 5. *The optimal strategy θ for an ACRA $(\mathbb{N})$ reachability game G can be computed in time $O\left(|Q| \, |\Sigma| \, 2^{|V| \log c_0 |Q|}\right)$, where c_0 is the largest constant appearing in the description of G.*

Note that the optimal strategy in ACRA $(\mathbb{N})$ games need not be memoryless: we might want to return to a state with a different register valuation. However, the strategy θ constructed in the proof of the above theorem is memoryless given the pair (q, val) of the current state and register valuation.

5.2 Hardness of solving ACRA ($\mathbb{D}$) reachability games

ACRA ($\mathbb{N}$) games are EXPTIME-hard We reduce the halting problem for linearly bounded alternating Turing machines to the problem of determining a winning strategy in an ACRA ($\mathbb{N}$) reachability game.

Definition 12. *A linearly bounded alternating Turing machine is a tuple* $M = (Q = Q_\vee \cup Q_\wedge, \Gamma, \delta, q_0, F, n)$*.* Q *is the state space which is partitioned into "or"-states* $Q_\vee$ *and and-states* $Q_\wedge$*.* $\Gamma = \{0, 1\}$ *is a binary tape alphabet, and* $\delta : Q \times \Gamma \times \{1, 2\} \to Q \times \Gamma \times \{L, R\}$ *is the transition function.* $q_0 \in Q$ *is the initial state, and* $F \subseteq Q$ *is the set of accepting states.* $n \in \mathbb{N}$ *is the length of the tape, specified in unary.*

The configuration is a tuple $\gamma = (q, \sigma, pos)$*, where* $q \in Q$ *is the current state,* $\sigma \in \Gamma^n$ *is the tape string, and* $pos \in \{1, 2, \ldots, n\}$ *is the position of the tape head. The initial configuration is* $(q_0, 0^n, 1)$*. In each configuration* (q, σ, pos)*,* δ *identifies two successors, corresponding to* $\delta(q, \sigma_{pos}, 1)$ *and* $\delta(q, \sigma_{pos}, 2)$ *respectively. Starting from a configuration* (q, σ, pos)*, the machine* M *eventually halts if either:*

1. $q \in F$ *is an accepting state, or*
2. $q \in Q_\vee$ *is an or-state and at least one of its successor configurations eventually halts, or*
3. $q \in Q_\wedge$ *is an and-state and both its successor configurations eventually halt.*

We construct the gadget shown in figure 5.1. There are two types of states: configuration states of the form (q, i, m) indicating that the TM is in state q, the tape head is in position i, and the last choice was move $m \in \{1, 2\}$, and challenge states of the form q^c_{ia} challenging the system to show that the symbol at position i of the tape is a. For each position i of the tape, we maintain two registers v_i, m_i. We maintain the invariant that $v_i = a = 1 - m_i$. Observe that to each state (q, i, m) and input symbol $a \in \Gamma$ indicating the current symbol under the head, there are two successors. If $q = q_\wedge$ is an and-state, then regardless of m', on processing (a, m'), either transition may be taken. If $q = q_\vee$ is an or-state, then on processing (a, m'), we transition to state (q', j, m'). Here q', j are respectively the next state and next tape head position. On each transition, the tape symbol registers v_i, m_i are appropriately updated. We now formalize:

Definition 13. *Let* $M = (Q = Q_\vee \cup Q_\wedge, \Gamma, \delta, q_0, F, n)$ *be a linearly bounded alternating Turing machine. Construct the following ACRA ($\mathbb{N}$) reachability game* $G_M = (Q', \Sigma, V, \delta', \mu, q'_0, F', \nu)$*.*

1. $Q' = \{q'_0\} \cup (Q \times [n] \times \{1, 2\}) \cup \left\{q^c_{i,a} \,\middle|\, \forall i \in [n], a \in \Gamma\right\}$*.*
2. $\Sigma = \Gamma \times \{1, 2\}$*.*
3. $V = \{v_i, m_i \mid \forall i \in [n]\} \cup \{z\}$*.*
4. *Define* δ' *as follows. For all symbols* $a \in \Sigma$*,* $(q'_0 \to^a (q_0, 1, 1)) \in \delta'$*.*
   (a) *Let* $q \in Q_\vee$*,* $a \in \Gamma$*,* $m, m' \in \{1, 2\}$*, and* $i \in [n]$*. Say that* $\delta(q, a, m) = (q', b, d)$*. If executing this transition with the tape head at* i *leads to it being at position* j*, then* $(q, i, m') \to^{(a,m)} (q', j, m) \in \delta'$ *and* $(q, i, m') \to^{(a,m)} q^c_{i,a} \in \delta'$*.*

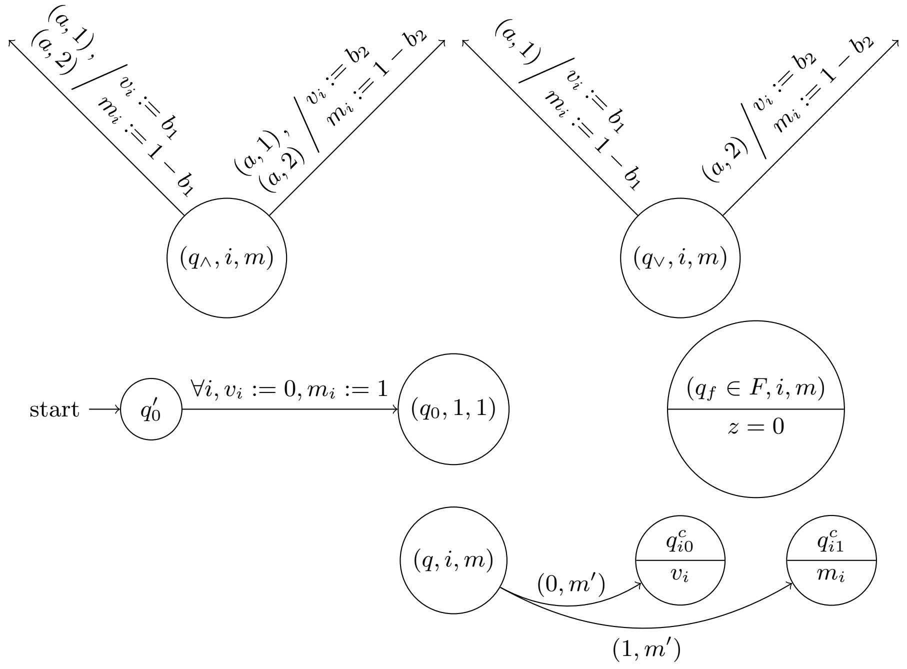


Fig. 5.1: Halting gadget G_M for a linearly bounded alternating Turing machine M.

   (b) Let $q \in Q_\wedge$, $a \in \Gamma$, $m, m', m'' \in \{1, 2\}$, and $i \in [n]$. Say that $\delta(q, a, m) = (q', b, d)$. Let j be the new head position, then $(q, i, m') \rightarrow^{(a, m'')} (q', j, m) \in \delta'$ and $(q, i, m') \rightarrow^{(a, m'')} q^c_{i,a} \in \delta'$.

5. *Define μ as follows. For all transitions $\tau = (q'_0 \rightarrow^a q) \in \delta$, $\mu(\tau, v_i) = (z, 0)$, and $\mu(\tau, m_i) = (z, 1)$, for all i. Let $\tau = (q, i, m) \rightarrow^{(a, m')} (q', j, m'')$ be some transition in δ'. Let b be the tape symbol left behind by $\delta(q, a, m'')$. Then define $\mu(\tau, v_i) = (z, b)$ and $\mu(\tau, m_i) = (z, 1 - b)$. For all other transitions $\tau \in \delta'$ and registers $v \in V$, define $\mu(\tau, v) = (v, 0)$.*
6. *Define $F' = \{(q, i, m) \mid q \in F\} \cup \left\{q^c_{i,0}, q^c_{i,1} \;\middle|\; 1 \leq i \leq n\right\}$. For all i, m, $\nu(q, i, m) = (z, 0)$. $\nu\left(q^c_{i,0}\right) = v_i$ and $\nu\left(q^c_{i,1}\right) = m_i$, for all i.*

Here $[n] = \{1, 2, \ldots, n\}$, and $Q_c = \left\{q^c_{i,a} \;\middle|\; \forall i, a\right\}$ are the challenge states. z is the constant register, always holding the value 0. By induction on the assertion that the starting configuration (q, σ, i) of the TM eventually halts, we have:

Claim. Let (q, σ, i) be a configuration starting from which M eventually halts. Then, for each m, there is a 0-winning strategy θ in G_M starting from $((q, i, m), val)$, where val encodes σ.

Let θ be a 0-winning strategy in G_M, and consider its strategy tree. At some internal node, let it issue input symbol (a_k, m_k), and let

$$\pi = q'_0 \rightarrow^{(a_0, m_0)} (q_1, i_1, n_1) \rightarrow^{(a_1, m_1)} \ldots \rightarrow^{(a_{k-1}, m_{k-1})} (q_k, i_k, m_k)$$

be the prefix of the run leading up to this node. It follows by induction on π that a_k is the current symbol under the tape head on the appropriate run of M (otherwise the adversary can lead the player to the challenge state $q^c_{i_k, a_k}$, but we assumed that θ was a 0-winning strategy). Since θ is 0-winning, every leaf of its decision tree must point to an accepting state. Furthermore, any winning strategy has to be associated with a finite decision tree, it follows that every run of M is accepting. Thus,

Claim. If there is a 0-winning strategy θ in G_M, then M eventually halts.

Note that Q' has $\Theta(|Q| n)$ elements, Σ has $\Theta(1)$ elements, and V has $\Theta(n)$ registers, where the tape size n was specified in unary. So G_M can be constructed in polynomial time given M. We thus conclude our argument:

Theorem 6. *Determining whether there is a winning strategy with budget k in an $ACRA(\mathbb{N})$ reachability game is* EXPTIME*-hard.*

Remark 3. Note that G_M never really needs to increment any register, since during all transitions, the values are either maintained unchanged, or reset from the constant register z. This suggests that the hardness comes from the combinatorial structure of the game rather than the specific grammar that allows increments.

Undecidability of ACRA $(\mathbb{Z})$ reachability games We reduce the halting problem for two-counter machines to the problem of solving a ACRA $(\mathbb{Z})$ reachability game. A two-counter machine M is a sequence of commands $L = \{l_1, l_2, \ldots, l_n\}$, where each command is of the form `inc` (c), `dec` (c), `if` $c \geq 0$ `goto` l_1 `else goto` l_2, or `halt`, where c refers to one of the counters $\{c_1, c_2\}$, and $l_1, l_2 \in L$ is the next location. Both counters are integer-valued and initialized to 0, and machine execution proceeds sequentially starting from location l_1. The semantics of these machines are standard, and we will not formally define them.

As with our earlier EXPTIME-hardness proof, the gadget G_M we construct has 4 registers v_1, m_1, v_2, m_2. Registers $v_1 = -m_1$ maintain the value of counter c_1, while registers $v_2 = -m_2$ maintain the value of counter c_2. The challenge states $q_{c<0}$, $q_{c\geq 0}$ for $c \in \{c_1, c_2\}$ force the system to prove the appropriate assertion about the counter value. The rest of the states are simply the locations L of the two-counter machine. In location $l \in L$, the system proposes the input symbol $(a, b) \in \{c_1 < 0, c_1 \geq 0\} \times \{c_2 < 0, c_2 \geq 0\}$. Each component of the tuple is an assertion about the value of the respective counter. Control proceeds to the next location l' depending on the location at l and the input symbol just received. The counters are incremented / decremented appropriately. We show that G_M has a 0-winning strategy θ iff M eventually halts.

Definition 14. *Let M be a two-counter machine. Then, the halting gadget $G_M = (Q, \Sigma, V, \delta, \mu, l_1, \nu)$ is the following ACRA $(\mathbb{Z})$ reachability game.*

1. *$Q = L \cup \{q_{c_1<0}, q_{c_1\geq 0}, q_{c_2<0}, q_{c_2\geq 0}\}$. We refer to the special states $Q_c = \{q_{c_1<0}, q_{c_1\geq 0}, q_{c_2<0}, q_{c_2\geq 0}\}$ as the challenge states.*
2. *$\Sigma = \{c_1 < 0, c_1 \geq 0\} \times \{c_2 < 0, c_2 \geq 0\}$.*
3. *$V = \{v_1, m_1, v_2, m_2, z\}$.*
4. *Define the transition relation δ as follows. Let l_i, l_j be arbitrary program locations so l_j can follow l_i in execution. Then*
   - *(a) Let l_i be either an increment or decrement instruction. Then $(l_i \rightarrow^a l_{i+1}) \in \delta$, for each $a \in \Sigma$.*
   - *(b) Let l_i be the instruction `if` $c_1 \geq 0$ `goto` l `else goto` l'. Then, the transitions $l_i \rightarrow^{(c_1\geq 0, a)} l$, $l_i \rightarrow^{(c_1\geq 0, a)} q_{c_1\geq 0}$, $l_i \rightarrow^{(c_1<0, a)} l'$, $l_i \rightarrow^{(c_1<0, a)} q_{c_1<0}$ occur in δ. And similarly for the conditional jumps on c_2.*
5. *Define the register update function μ as follows. Let l_i be instruction `inc` (c_1). Then $\mu(l_i \rightarrow^a l_{i+1}, v_1) = (v_1, 1)$, and $\mu(l_i \rightarrow^a l_{i+1}, m_1) = (m_1, -1)$. On `dec` (c_1), v_1 is decremented and m_1 is incremented. Similarly for `inc` (c_2) and `dec` (c_2). In all other transitions τ, $\mu(\tau, v) = (v, 0)$, for each register v, i.e. the register is left unchanged.*
6. *For all halting locations $l =$ `halt` $\in L$, define $\nu(l) = (z, 0)$. For all non-halting locations $l \in L$, define $\nu(l) = \perp$. For the challenge states, define $\nu(q_{c_i<0}) = v_i + 1$, $\nu(q_{c_i\geq 0}) = m_i$.*

Theorem 7. *Determining whether there is a winning strategy with budget k in an ACRA $(\mathbb{Z})$ reachability game is undecidable.*

Proof. We establish the claim that M halts iff G_M permits a winning strategy γ with budget 0. The undecidability of solving ACRA $(\mathbb{Z})$ reachability games follows from the undecidability of the halting problem for two-counter machines [12].

First, assume that M halts. We want to construct a winning strategy γ with budget 0. Consider the finite execution of M. After executing the first k steps, the player issues the symbol (s_1, s_2), where s_1, s_2 are respectively the signs of the values in c_1, c_2 after the two-counter machine executes for k steps. That this is a 0-budget strategy follows from the invariant that after the first k steps, there is only one run that does not end in a challenge state, and in that run, v_i holds the value of c_i, and $m_i = -v_i$.

Conversely, assume that a winning strategy γ exists. Then the decision tree encoding the strategy has to be finite. In any such strategy tree, challenge states may appear only at the leaves. Observe that any challenge state $q_{c<0}$ or $q_{c\geq 0}$ has a sibling state $l \in L$. Furthermore, for all non-halting locations $l \neq \texttt{halt}$, $\nu(l) = \perp$, and so no non-halting location can be at the leaf of the strategy tree. Thus, some leaf of the strategy tree has to be in a location $l = \texttt{halt}$. Consider the finite sequence of input symbols leading to this location. Because it is a winning strategy, at each node of the tree, if the next input symbol is (s_1, s_2), then s_1, s_2 are respectively the signs of the values in c_1, c_2 after the machine executes for the appropriate number of steps. Thus, this trace encodes a halting run of the machine.

6 Conclusion

In this paper, we studied two decision problems for additive regular functions: determining the register complexity, and alternating reachability in ACRAs. The register complexity is the largest number k so that every ACRA implementing f has at least k registers. We developed an abstract characterization of register complexity as separability and showed that computing it is PSPACE-complete. We then studied the reachability problem in alternating ACRAs, and showed that it is undecidable for ACRA $(\mathbb{Z})$ and EXPTIME-complete for ACRA $(\mathbb{N})$ games. Future work includes proving similar characterizations and providing algorithms for register minimization in more general models such as streaming string transducers. String concatenation does not form a commutative monoid, and the present paper is restricted to unary operators (increment by constant), and so the technique does not immediately carry over. Another interesting question is to find a machine-independent characterization of regular functions $f : \Sigma^* \to \mathbb{Z}_\perp$. A third direction of work would be extending these ideas to trees and studying their connection to alternating ACRAs.

References

1. Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *38th Annual Symposium on Principles of Programming Languages*, pages 599–610, 2011.

2. Rajeev Alur and Loris D'Antoni. Streaming tree transducers. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 42–53. Springer, 2012.
3. Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. *To appear in the 28th Annual Symposium on Logic in Computer Science*, Full version available at http://www.cis.upenn.edu/~alur/rca12.pdf, 2013.
4. Benjamin Aminof, Orna Kupferman, and Robby Lampert. Rigorous approximated determinization of weighted automata. In *Proceedings of the 20th Annual Symposium on Logic in Computer Science*, pages 345–354, June 2011.
5. Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata with group actions. In *26th Annual Symposium on Logic in Computer Science*, pages 355–364, 2011.
6. Krishnendu Chatterjee, Laurent Doyen, and Thomas Henzinger. Quantitative languages. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2008.
7. John Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation.* Prentice Hall, 3rd edition, 2006.
8. Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
9. Daniel Kirsten. Decidability, undecidability, and PSPACE-completeness of the twins property in the tropical semiring. *Theoretical Computer Science*, 420:56–63, February 2012.
10. Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, 1977.*, pages 254–266, 31 Oct – 2 Nov 1977.
11. Nicolas Markey. Weighted automata: Model checking and games. *Lecture notes*, Available at http://www.lsv.ens-cachan.fr/~markey/Teaching/MPRI/2008-2009/MPRI-2.8b-4.pdf, 2008.
12. Marvin Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, 1967.
13. Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, 2000.
14. Mehryar Mohri. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 213–254. Springer, 2009.
15. Christos Papadimitriou and Mihalis Yannakakis. Multiobjective query optimization. In *Proceedings of the 20th Symposium on Principles of Database Systems*, PODS '01, pages 52–59. ACM, 2001.